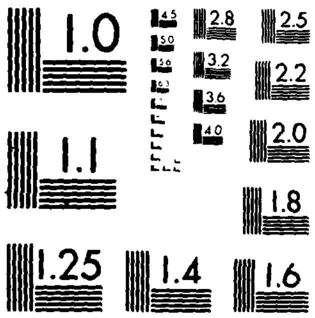
1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

NRL [illegible]

NRL Memorandum Report [illegible]

A-7E Software Module Guide

K. H. Britton and D. L. Parnas

Computer Science and Systems Branch
Information Technology Division

December 8, 1981

NAVAL RESEARCH LABORATORY
Washington, D.C.

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Memorandum Report 4702	2. GOVT ACCESSION NO. AD-A110 446	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A-7E SOFTWARE MODULE GUIDE		5. TYPE OF REPORT & PERIOD COVERED Interim report on continuing NRL problem.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) K. H. Britton and D. L. Parnas		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62721N; SF21242101; 75-0106-0-2
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		12. REPORT DATE December 8, 1981
		13. NUMBER OF PAGES 35
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Specifications
Real time systems
Software documentation
Abstract interfaces
Modules
Information hiding
Avionics software

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document describes the basic organization of NRL's version of the A-7E onboard flight software. The report describes a structure in which modules have been designed in accordance with the Information Hiding Principle. Because of the large number of modules that result when this principle is applied, the modules are arranged in a hierarchy. The hierarchy should simplify the task of a maintenance programmer assigned to make a specific modification. The document also describes the principles used in the design of the software. It is intended to be useful both as a guide to the A-7E software and as a model for those developing other software systems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

A-7E SOFTWARE MODULE GUIDE

I. INTRODUCTION

PURPOSE

The A-7E Module Guide describes the module structure of the A-7E flight software produced by the Naval Research Laboratory. It provides an orientation for software engineers who are new to the A-7E system, explains the principles used to design the structure, and shows how responsibilities are allocated among the major modules.

This guide is intended to lead a reader to the module that implements a particular aspect of the system. It states the criteria used to assign a particular responsibility to a module and arranges the modules in such a way that a reader can find the information relevant to his purpose without searching through unrelated documentation.

The module guide should be read before any other design documentation for the NRL A-7E software, because the guide defines the scope and contents of the individual design documents.

This guide describes and prescribes the module structure. Changes in the structure will be promulgated as changes to this document. Changes are not official until they appear in that form. This guide is a rationalization of the structure, not a description of the design process that led to it.

PREREQUISITE KNOWLEDGE

Readers are assumed to be familiar with the terminology and organization of Software Requirements for the A-7E Aircraft [1], which will be referred to as "the requirements document". They should have a general idea of the functions performed by the A-7E flight software, know something about avionics computers such as the IBM 4 PI TC-2, and be familiar with the types of hardware devices that are connected to such computers. These are described in adequate detail in the requirements document.

ORGANIZATION

Section II gives the background for the design. It states 1) the goals that motivated the module design decisions presented in this document; 2) the basic principles on which the design is based; and 3) the relationship between the module structure and two other structures of the NRL A-7E software.

Section III, the main body of the document, presents a hierarchical decomposition of the software into top-level, second-level, and third-level modules. The modules at each level are components of modules of the next higher level.

Terms, such as "module", that are used with a special meaning in NRL's A-7E design, are defined in the glossary. Readers who are not familiar with the NRL A-7E project's terminology should study the glossary before reading further.

Manuscript submitted October 23, 1981.

II. BACKGROUND

THE A-7E SOFTWARE STRUCTURES

A structural description of a software system shows the program's decomposition into parts and the relations between those parts. A-7E programmers must be concerned with three structures: (a) the module structure, (b) the uses structure, and (c) the process structure. This section contrasts these structures.

(a) A module is a work assignment for a programmer or programmer team. Each module consists of a group of closely related programs. The module structure is the decomposition of the program into modules and the assumptions that the team responsible for each module is allowed to make about the other modules.

(b) In the uses structure the components are programs, i.e., not modules but parts of modules; the relation is "requires the presence of". The uses structure determines the executable subsets of the software [8]. Guidelines for the design of the A-7E uses structure are given in [15].

(c) The process structure is a decomposition of the run-time activities of the system into units known as processes. Processes are not programs; there is no simple relation between modules and processes. The implementation of some modules may include one or more processes, and any process may invoke programs in several modules. Guidelines for the A-7E process design are given in [7].

The rest of this document describes the module structure.

GOALS OF THE A-7E MODULE STRUCTURE

The overall goal of the decomposition into modules is reduction of software cost by allowing modules to be designed and revised independently. Specific goals of the module decomposition are:

(a) each module's structure should be simple enough that it can be understood fully;

(b) it should be possible to change the implementation of one module without knowledge of the implementation of other modules and without affecting the behavior of other modules;

(c) the ease of making a change in the design should bear a reasonable relationship to the likelihood of the change being needed; it should be possible to make likely changes without changing any module interfaces; less likely changes may involve interface changes, but only for modules that are small and not widely used. Only very unlikely changes should require changes in the interfaces of widely used modules;

(d) it should be possible to make a major software change as a set of independent changes to individual modules, i.e., except for interface changes, programmers changing the individual modules should not need to communicate. If the interfaces of the modules are not revised, it should be possible to run and test any combination of old and new module versions.

As a consequence of the goals above, the A-7E software is composed of many small modules. They have been organized into a tree-structured hierarchy; each nonterminal node in the tree represents a module that is composed of the modules represented by its descendents. The hierarchy is intended to achieve the following additional goals:

(e) A software engineer should be able to understand the responsibility of a module without understanding the module's internal design.

(f) A reader with a well defined concern should easily be able to identify the relevant modules without studying irrelevant modules. This implies that the reader be able to distinguish relevant modules from irrelevant modules without looking at their components.

DESIGN PRINCIPLE

The A-7E module structure is based on the decomposition criteria known as information hiding [2]. According to this principle, system details that are likely to change independently should be the secrets of separate modules; the only assumptions that should appear in the interfaces between modules are those that are considered unlikely to change. Every data structure is private to one module; it may be directly accessed by one or more programs within the module but not by programs outside the module. Any other program that requires information stored in a module's data structures must obtain it by calling module programs.

Applying this principle is not always easy. It is an attempt to minimize the expected cost of software and requires that the designer estimate the likelihood of changes. Such estimates are based on past experience, and may require knowledge of the application area, as well as an understanding of hardware and software technology.

In a few cases information that is likely to change must be communicated between modules. To reduce the cost of software changes, use of some modules or portions of a module interface, may be restricted. Restricted interfaces are indicated by "(R)" in the documentation. Often the existence of certain smaller modules is itself a secret of a larger module. In a few cases, we have mentioned such modules in this document in order to clearly specify where certain functions are performed. Those modules are referred to as hiddden modules and indicated by "(H)" in the documentation.

MODULE DESCRIPTION

Three ways to describe a module structure based on information-hiding are: (1) by the roles played by the individual modules in the overall system operation; (2) by the secrets associated with each module; and (3) by the facilities provided by each module. This document describes the module structure by characterizing each module's secrets. Where useful, we also include a brief description of the role of the module. The description of facilities is relegated to the module specifications (e.g. [3]).

For some modules we find it useful to distinguish between a primary secret, which is hidden information that was specified to the software designer, and a secondary secret, which refers to implementation decisions made by the designer when implementing the module designed to hide the primary secret.

Although we have attempted to make the decomposition rules as precise as possible, the possibility of future changes in technology makes some of the boundaries fuzzy. Some sections point out fuzzy areas and discuss additional information used to resolve ambiguities.

MODULE INITIALIZATION

Every module in the A-7E software can contain variables that must be given initial values when the computer is turned on. Each module contains a program for initialization, (denoted +module ID_INIT+), that will be called when power up occurs. There will be a main initialization program that is invoked at power up. It will invoke the INIT programs for the three top-level modules. The INIT program for a module will call the INIT programs for each of its submodules. These programs will be executed, sequentially, before any of the parallel processes begin to execute.

FUTURE ADDITIONS TO MODULES

It is often the case that a particular version of the system may not need features of a module that are likely to be needed in other versions. Where we have identified features that may be needed in future versions but are not needed in the initial version, they are included in the module interface descriptions but it is noted that they will not be needed in the initial version. The programmer can use this information about likely future additions to design his software for easier extension [8].

TREATMENT OF UNDESIRED EVENTS

Development versions of all modules will check for and report undesired events (UEs). Each module interface description contains a list of possible UEs. In general, such a list should constitute a classification of all the things that might go wrong. It should include hardware errors, software errors and errors caused by the using program. However, much of the UE detection and correction must be removed from the production version of the system because of space limitations. In the NRL A-7E context, conditions that must be handled by the initial production version will not be considered UEs. For a more complete discussion of UEs see [9].

The remainder of this report provides a top-down overview of the module structure.

III. A-7E MODULE STRUCTURE

A: TOP-LEVEL DECOMPOSITION

The software system consists of the three modules described below.

A:1 HARDWARE-HIDING MODULE

The Hardware-Hiding Module includes the programs that need to be changed if any part of the hardware is replaced by a new unit with a different hardware/software interface but with the same general capabilities. This module implements virtual hardware that is used by the rest of the software. The primary secrets of this module are the hardware/software interfaces described in chapters 1 and 2 of the requirements document. The secondary secrets of this module are the data structures and algorithms used to implement the virtual hardware.

A:2 BEHAVIOR-HIDING MODULE

The Behavior-Hiding Module includes programs that need to be changed if there are changes in the sections of the requirements document that describe the required behavior (chapters 3 and 4). The content of those sections is the primary secret of this module. These programs determine the values to be sent to the virtual output devices provided by the Hardware-Hiding Module.

A:3 SOFTWARE DECISION MODULE

The Software Decision Module hides software design decisions that are based upon mathematical theorems, physical facts, and programming considerations such as algorithmic efficiency and accuracy. The secrets of this module are <u>not</u> described in the requirements document. This module differs from the other modules in that both the secrets and the interfaces are determined by software designers. Changes in these modules are more likely to be motivated by a desire to improve performance than by externally imposed changes.

Notes on the top-level decomposition:

Fuzziness in the above classifications is unavoidable for the following reasons:

(a) The line between requirements definition and software design has been determined in part by decisions made when the requirements documents are written; for example, weapon trajectory models may be chosen by system analysts and specified in the requirements document, or they may be left to the discretion of the software designers.

(b) The line between hardware characteristics and software design may vary. Hardware can be built to perform some of the services currently performed by the software; consequently, certain modules can be viewed either as modules that hide hardware characteristics or as modules that hide software design decisions.

(c) Changes in the hardware or in the behavior of the system or its users may make a software design decision less appropriate.

(d) All software modules include software design decisions; changes in any module may be motivated by efficiency or accuracy considerations.

To reduce the fuzziness, we have based our decomposition on the current requirements document. In particular,

(a) The line between requirements and software design is defined by our requirements document. When the requirements document specifies an algorithm, we do not consider the design of the algorithm to be a software design decision. If the requirements document only states requirements that the algorithm must meet, we consider the program that implements that algorithm to be part of a Software Decision Module.

(b) The line between hardware characteristics and software design is based on estimates of the likelihood of future changes. For example, if it is reasonably likely that future hardware will implement a particular facility, the software module that implements that facility is classified as a hardware-hiding module; otherwise, the module is considered a software design module. In most cases we have taken a conservative stance; the design is based on the assumption that drastic changes are less likely than evolutionary changes.

(c) A module is included in the Software Decision Module only if it would remain correct, albeit less efficient, when there are changes in the requirements document.

(d) A module will be included in the software decision category only if its secrets do not include information documented in the software requirements document.

B: SECOND LEVEL DECOMPOSITION

B:1 HARDWARE-HIDING MODULE DECOMPOSITION:

The Hardware Hiding Module comprises two modules.

B:1.1 EXTENDED COMPUTER MODULE

The Extended Computer Module hides those characteristics of the hardware/software interface of the avionics computer that we consider likely to change if the computer is modified or replaced.

Avionics computers differ greatly in their hardware/software interfaces and in the capabilities that are implemented directly in the hardware. For example, some avionics computers include a hardware approximation of real numbers, while others perform approximate real number operations by a programmed sequence of fixed-point operations. Some avionics systems include a single processor; some systems provide several processors. The Extended Computer provides an instruction set that can be implemented efficiently on most avionics computers. This instruction set includes the operations on application-independent data types, sequence control operations, and general I/O operations.

The primary secrets of the Extended Computer are: the number of processors, the instruction set of the computer, and the computer's capacity for performing concurrent operations.

The structure of the Extended Computer Module is given in section C:1.1. Specifications for Extended Computer submodules are given in [4].

B:1.2 DEVICE INTERFACE MODULE

The Device Interface Module hides the characteristics of the peripheral devices that are considered likely to change. Each device might be replaced by an improved device capable of accomplishing the same tasks. Replacement devices differ widely in their hardware/software interfaces. For example, all angle-of-attack sensors measure angle-of-attack, but they differ in input format, timing, and the amount of noise in the data.

The Device Interface Module provides virtual devices to be used by the rest of the software. The virtual devices do not necessarily correspond to physical devices because all of the hardware providing a capability is not necessarily in one physical unit. Further, there are some capabilities of a physical unit that are likely to change independently of others; it is advantageous to hide characteristics that may change independently in different modules.

The primary secrets of the Device Interface Module are those characteristics of the present devices documented in the requirements document and not likely to be be shared by replacement devices.

The structure of the Device Interface Module is given in section C:1.2. Specifications for Device Interface submodules are given in [3].

Notes on the Hardware-Hiding Module Decomposition

Our distinction between computer and device is based on the current hardware and is the one made in the requirements document. Information that applies to more than one device is considered a secret of the Extended Computer; information that is only relevant to one device is a secret of a Device Interface Module. For example, there is an analog to digital converter that is used for communicating with several devices; it is hidden by the Extended Computer although it could be viewed as an external device. As another example, there are special outputs for testing the I/O channels; they are not associated with a single device. These too are hidden by the Extended Computer.

If all the hardware were replaced simultaneously, there might be a significant shift in responsibilities between computer and devices. In systems like the A-7E such changes are unusual; the replacement of individual devices or the replacement of the computer alone is more likely. Our design is based on the expectation that this pattern of replacement will continue to hold.

B:2 BEHAVIOR-HIDING MODULE DECOMPOSITION:

The Behavior Hiding Module consists of 2 modules: a Function Driver (FD) Module supported by a Shared Services (SS) Module.

B:2.1 FUNCTION DRIVER MODULE

The Function Driver Module consists of a set of individual modules called Function Drivers; each Function Driver is the sole controller of a set of closely related outputs. The outputs are either part of a virtual device or provided by the Extended Computer for test purposes. The primary secrets of the Function Driver Module are the rules determining the values of these outputs.

The structure of the Function Driver Module is given in section C:2.1. Specifications for the Function Driver Module are found in [9].

B:2.2 SHARED SERVICES MODULE

Because all the Function Drivers control systems in the same aircraft, some aspects of the behavior are common to several Function Drivers. We expect that if there is a change in that aspect of the behavior, it will affect all of the functions that share it. Consequently we have identified a set of modules, each of which hides an aspect of the behavior that applies to two or more of the outputs.

The structure of the Shared Services Module is found in section C:2.2. Specifications for the Shared Services Module are found in [10].

Notes on the Behavior-Hiding Module structure

Because users of the documentation cannot be expected to know which aspects of a function's behavior are shared, the documentation for the Function Driver Modules will include a reference to the Shared Services Modules that it uses. A maintenance programmer should always begin his inquiry with the appropriate function driver. He will be directed to the Shared Services Modules when appropriate.

B:3 SOFTWARE DECISION MODULE DECOMPOSITION

The Software Decision Module has been divided into (1) the Application Data Type Module, which hides the implementation of certain variables, (2) the Physical Model Module, which hides algorithms that simulate physical phenomena, (3) the Data Banker Module, which hides the data-updating policies, (4) the System Generation Module, which hides decisions that are postponed until system generation time, (5) the Software Utility Module, which hides algorithms that are used in several other modules, and (6) the Resource Monitor Module, which hides the synchronization needed for resources that are shared by several processes.

B:3.1 APPLICATION DATA TYPE MODULE

The Application Data Type Module supplements the data types provided by the Extended Computer Module with data types that are particularly useful for avionics applications and do not require a computer dependent implementation. These data types are implemented using the data types provided by the Extended Computer; variables of those types are used just as if the types were built into the Extended Computer.

The secrets of the Application Data Type Module are the data representation used in the variables and the programs used to implement operations on those variables. These variables can be used without consideration of units. Where necessary, the modules provide conversion operators, which deliver or accept real values in specified units.

Run-time efficiency considerations sometimes dictate that an implementation of an application data type be based on a secret of another module. In that case, the data type will be specified in the Application Data Type Module documentation, but the implementation will be described in the documentation of the other module. The Application Data Type Module documentation will contain the appropriate references in such cases.

The structure of the Application Data Type Module is given in section C:3.1. Specifications for these modules may be found in [11].

B:3.2 PHYSICAL MODEL MODULE

The software requires estimates of quantities that cannot be measured directly but can be computed from observables using models of the physical world. The primary secrets of the Physical Model Module are the physical models; the secondary secrets are the computer implementations of those models.

The structure of the Physical Model Module is given in section C:3.2. Interface specifications are found in [13].

B:3.3 DATA BANKER MODULE

Most data are produced by one module and consumed by another. In many cases, the consumers should receive a value as up-to-date as practical. The Data Banker Module acts as a "middleman" and determines when new values for these data are computed. The Data Banker obtains values from producers; consumer programs obtain data from Data Banker access programs. The producer and consumers of a particular datum can be written without knowing whether or not the Data Banker stores the value or when a stored value is updated. In most cases, neither the producer nor the consumer need be modified if the updating policy changes.

The Data Banker is not used if consumers require specific members of the sequence of values computed by the producer or if they require values associated with a specific time such as the moment when an event occurs. There are very few instances of this in the A-7E software.

Some of the updating policies that can be implemented in the Data Banker are described in the following table, which indicates whether or not the Data Banker stores a copy of the item and when a new value is computed.

<u>Name</u>	<u>Storage</u>	<u>When new value produced</u>
<u>on demand:</u>	No	Whenever a consumer requests the value
<u>periodic:</u>	Yes	Periodically. Consumers get the most recently stored value.
<u>event driven:</u>	Yes	Whenever the data banker is notified, by the occurrence of an event, that the value may have changed. Consumers get the most recently stored value.
<u>conditional:</u>	Yes	Whenever a consumer requests the value, provided certain conditions are true. Otherwise, a previously stored value is delivered.
<u>producer fed:</u>	Yes	Determined by the producer, which calls an access routine from the data banker to store a value. Consumers get the most recently stored value.

The choice among these and other updating policies should be based on the consumers' accuracy requirements, how often consumers require the value, the maximum wait that consumers can accept, how often the value changes, and the cost of producing a new value. Since the decision is not based on coding details of either consumer or producer, it is usually not necessary to rewrite a Data Banker Module when producer or consumer change.

The Data Banker is used regardless of the frequency of the updates; for example, it provides values of data received through the panel, even though they are rarely updated. The Data Banker is used as long as consumer and producer are separate modules, even when they are both submodules of a larger module.

Specifications of the interface to the Data Banker Module can be found in [6].

B:3.4 SYSTEM GENERATION MODULE

The primary secrets of the System Generation Module are decisions that are postponed until system-generation time. These include the values of system generation parameters and the choice among alternative implementations of a module. The secondary secrets of the System Generation Module are the method used to generate a machine-executable form of the code and the representation of the postponed decisions. Most of the programs in this module do not run on the on-board computer; they run on the computer used to generate the code for the on-board system.

The structure of the System Generation Module is given in section C:3.4. Specifications for the System Generation Modules may be found in [14].

B:3.5 SOFTWARE UTILITY MODULE

The primary secrets of this module are the algorithms implementing common software functions such as start process, and mathematical functions such as square-root and logarithm.

B:3.6 RESOURCE MONITOR MODULE

The primary secrets of this module are the synchronization algorithms used to provide orderly access to shared resources such as devices and data structures. Programs from these modules are used within modules that provide virtual resources.

C: THIRD-LEVEL DECOMPOSITION

C:1.1 EXTENDED COMPUTER MODULE DECOMPOSITION

C:1.1.1 DATA TYPE MODULE

The Data Type Module implements variables and operators for real numbers, time periods, and bit strings. The data representations and data manipulation instructions built into the computer hardware are the primary secret of this module.

C:1.1.2 DATA STRUCTURE MODULE

The Data Structure Module allows user programs to create arrays of EC variables and to access elements within them. The representation of these arrays in physical memory and the implementation of array access operations are the secrets of this module.

C:1.1.3 INPUT/OUTPUT MODULE

The Input/Output Module transmits bit strings to and from peripheral devices without any interpretation of the values. The computer instructions available to transmit and receive bitstrings are the primary secret of this module.

C:1.1.4 COMPUTER STATE MODULE

The Computer State Module keeps track of the current state of the Extended Computer, which can be either operating, off, or failed, and signals relevant state changes to user programs. The primary secret is the way that the hardware detects and causes state changes. After the EC has been properly initialized, this module signals the event that starts the initialization for the rest of the software.

C:1.1.5 PARALLELISM CONTROL MODULE

The virtual computer provided by the Extended Computer Module executes a set of processes in parallel. The Parallelism Control Module determines the rate of progress of processes, given the constraints imposed by synchronization operations and by the timing parameters in the process definitions. The synchronization operations and scheduler are part of this module.

The number of processors and the mechanism for process switching are the primary secrets of this module. The representation of processes, allocation policy for the processors, and the algorithms implementing synchronization operations are secondary secrets.

C:1.1.6 SEQUENCE CONTROL MODULE

The Sequence Control Module determines the order of statement execution within a process. It provides an instruction that permits loops and conditional selection among alternative code sections; it also provides subprogram invocation and timing mechanisms.

The primary secrets of this module are the sequence control and timer mechanisms of the actual computer.

C:1.1.7 DIAGNOSTICS MODULE (R)

The Diagnostics Module provides diagnostic programs to test the interrupt hardware, the I/O hardware, and the memory. Use of this module is restricted because the information it returns may reveal secrets of the Extended Computer, i.e., programs that use it may have to be revised if the avionics computer is replaced by another computer.

C:1.1.8 VIRTUAL MEMORY MODULE (H)

The Virtual Memory Module presents a uniformly addressable virtual memory to the other Extended Computer submodules, allowing them to use virtual addresses for both data and subprograms. The primary secrets of the Virtual Memory Module are the hardware addressing methods for data and instructions in real memory; differences in the way that different areas of memory are addressed are hidden. The secondary secrets of the module are the policy for allocating real memory to virtual addresses and the programs that translate from virtual address references to real instruction sequences.

This module is invisible to Extended Computer users; the Extended Computer offers a "typed" memory instead of a memory consisting of general-purpose words. The memory provided by the Virtual Memory Module can contain both program and data, but the Extended Computer has no operations that allow access to programs as data.

C:1.1.9 INTERRUPT HANDLER MODULE (H)

The Interrupt Handler Module is responsible for responding to external interrupts, interpreting them, and reporting them; to user programs, an interrupt event looks like any other event signalled by the software. As a result, users are not aware of either interrupts or the Interrupt Handler Module.

The primary secrets of this Module are the way that the hardware behaves when an interrupt occurs, the mapping between hardware interrupts and events signalled by the software, and the method used to identify the source of the interrupt. The secondary secret is the mechanism for translating hardware interrupts into software events.

C:1.2 DEVICE INTERFACE MODULE DECOMPOSITION

The following table describes the Device Interface submodules (DIMs)and their secrets. The phrase "How to read.. " is intended to be interpreted quite liberally, e.g., it includes device-dependent corrections, filtering and any other actions that may be necessary to determine the physical value from the device input. All of the DIMs hide the procedures for testing the device that they control.

Section	Virtual Device	Secret: How to . . .
C:1.2.1	AIR DATA COMPUTER	read barometric altitude, true airspeed, and Mach number.
C:1.2.2	ANGLE OF ATTACK SENSOR	read angle of attack.
C:1.2.3	AUDIBLE SIGNAL DEVICE	cause an audible tone in the cockpit to be on, off, or beeping.
C:1.2.4	COMPUTER FAIL DEVICE	signal computer failure to the pilot and several devices.
C:1.2.5	DOPPLER RADAR SET	read ground speed and drift angle.
C:1.2.6	FLIGHT INFORMATION DISPLAYS	display an azimuth and an elevation displacement, two points on a circle relative to a fixed reference point, and an unsigned decimal number.
C:1.2.7	FORWARD LOOKING RADAR	measure slant range distance to a point on the ground; display a point on a radar screen.
C:1.2.8	HEAD-UP DISPLAY	display symbols in the pilot's field of vision.
C:1.2.9	INERTIAL MEASUREMENT SET	measure aircraft attitude, heading, and velocity; adjust the orientation of the platform axes.
C:1.2.10	PANEL	display and accept information in the form of digits and letters.
C:1.2.11	PROJECTED MAP DISPLAY SET	position and orient a map display.
C:1.2.12	RADAR ALTIMETER	read the altitude of the aircraft above local ground or water level.

Section	Virtual Device	Secret: How to . . .
C:1.2.13	SHIPBOARD INERTIAL NAVIGATION SYSTEM	read the position, attitude and velocity of a nearby ship carrying SINS transmission equipment.
C:1.2.14	SLEW CONTROL	read data from a device indicating a two dimensional displacement from an origin.
C:1.2.15	SWITCH BANK	read the positions of all switches that do not affect other hardware devices.
C:1.2.16	TACAN	read bearing and slant range to a TACAN station.
C:1.2.17	VISUAL INDICATORS	cause visible indicators to be on, blinking or off.
C:1.2.18	WAYPOINT INFORMATION SYSTEM	read received data giving the positions of waypoints.
C:1.2.19	WEAPON CHARACTERISTICS	determine specific characteristics of various weapon types.
C:1.2.20	WEAPON RELEASE SYSTEM	ascertain weapon release actions the pilot has requested; cause weapons to be prepared and released.
C:1.2.21	WEIGHT ON GEAR	tell if the plane is resting on the landing gear.

C:2.1 FUNCTION DRIVER MODULE DECOMPOSITION

The following table describes the Function Driver submodules and their secrets.

Section	Function Driver	Secret
C:2.1.1	AIR DATA COMPUTER FUNCTIONS	What must be done to initialize the virtual Air Data Computer.
C:2.1.2	AUDIBLE SIGNAL FUNCTIONS	When the audible signal should be on, off, or beeping.
C:2.1.3	COMPUTER FAIL SIGNAL FUNCTIONS	When to signal computer failure.
C:2.1.4	DOPPLER RADAR FUNCTIONS	When to start and stop the Doppler Radar.
C:2.1.5	FLIGHT INFORMATION DISPLAY FUNCTIONS	What information should be displayed.
C:2.1.6	FORWARD LOOKING RADAR FUNCTIONS	What the current FLR mode should be. Where to position the cursors in the FLR display. Where to aim the FLR.
C:2.1.7	HEAD-UP DISPLAY FUNCTIONS	Where the movable HUD symbols should be placed. Whether a HUD symbol should be on, off or blinking. What information should be displayed on the fixed-position displays.
C:2.1.8	INERTIAL MEASUREMENT SET FUNCTIONS	Rules determining the scale to be used for the IMS velocity measurements. When to initialize the velocity measurements. How much to rotate the IMS for alignment.
C:2.1.9	PANEL FUNCTIONS	What information should be displayed on panel windows. When the enter light should be turned on.

Section	Function Driver	Secret
C:2.1.10	PROJECTED MAP DISPLAY SET FUNCTIONS	What geographical location should be displayed on the map. How the map should be oriented. Where the map indicators should be positioned.
C:2.1.11	SINS FUNCTIONS	When to start and stop SINS reception.
C:2.1.12	VISUAL INDICATOR FUNCTIONS	When the visual indicators should be on, off, or blinking.
C:2.1.13	WEAPON RELEASE FUNCTIONS	When to prepare and release a Weapon.
C:2.1.14	GROUND TEST FUNCTIONS	When and how to use the EC test outputs.

C:2.2 SHARED SERVICES MODULE DECOMPOSITION

The Shared Services Module comprises the following modules.

C:2.2.1 MODE DETERMINATION MODULE

The Mode Determination Module determines system modes (as defined in the requirements document). It signals the occurrence of mode transitions and makes the identity of the current modes available. The primary secrets of the Mode Determination Module are the mode transition tables in the requirements document.

C:2.2.2 STAGE DIRECTOR MODULE

In some modes, the system sequences through stages; in each stage the program is trying to achieve a goal and the end of each stage is marked by the achievement of that goal. Whether or not a goal has been achieved is determined by the program itself, rather than by an external event. Although many of the stages occur in several modes, the modes differ in the definition of the goals and the sequence of stages.

There are stage directors for each of the alignment modes and for the ground test mode. The primary secret of each Stage Director Module is the sequence of stages and the predicates that determine when a stage transition occurs. The behavior required in the individual stages is a secret of Function Driver submodules, but the rules determining when to proceed from one stage to the next are hidden in the Stage Director Module.

C:2.2.3 SHARED SUBROUTINE MODULE

The Shared Subroutine Modules hide parts of the function definitions that are shared, perhaps with some modification, by several functions. Where, in our judgment, the sharing is not a coincidence and a change in one function driver is likely to be accompanied by a similar change in the others, the routines have been included in the Shared Subroutine module to avoid duplication of code and documentation. In some cases, we have made sharing possible by parameterization.

C:2.2.4 SYSTEM VALUE MODULE

A System Value submodule computes a set of values, some of which are used by more than one Function Driver. The secrets of a System Value submodule are the rules in the requirements that define the values that it computes. The shared rules in the requirements specify such things as 1) selection among several alternative sources, 2) applying filters to values produced by other modules, or 3) imposing limits on a value calculated elsewhere.

This module may include a value that is only used in one Function Driver if the rule used to calculate that value is the same as that used to calculate other shared values.

Each System Value submodule is also responsible for signaling events that are defined in terms of the values it computes.

C:2.2.5 PANEL I/O SUPPORT MODULE

The Panel I/O Support Module provides formatting services for the Function Drivers that display and accept data through the panel. The primary secrets are the required data display and input formats. This module will not be the one that hides the hardware/software interface of a particular panel; it will hide characteristics of the virtual panel created by the Device Interface Module, providing a more convenient interface. This module also signals events about panel operations.

C:2.2.6 DIAGNOSTIC I/O SUPPORT MODULE (R)

The Diagnostic I/O Support Module contains the programs that convert hardware-dependent diagnostic information to the format required for display on one of the virtual devices provided by the Device Interface Modules. The primary secrets of this module are how to get the information to be displayed and the format in which it is to be displayed. This module may have to be changed if either the hardware supplying the information or the format requirements change.

C:2.2.7 EVENT TAILORING MODULE

This module contains programs that detect and signal changes in conditions that appear in the definitions of the behavior-hiding modules. Although the values that determine these conditions may be computed in other modules, the exact definitions of the conditions may change if the requirements change. These modules may be thought of as tailoring the generally applicable (requirements independent) events and access programs provided by other modules to the specific needs of the current requirements. The other modules can be implemented without any knowledge that these events are of significance. The primary secrets of these modules are the definitions of the events of interest.

C:3.1 APPLICATION DATA TYPE MODULE DECOMPOSITION

The Application Data Type Module is divided into three submodules.

C:3.1.1 SYSTEM DATA TYPE MODULE

The System Data Type Module implements variables of the following widely used types: accelerations, angles, angular rates, character literals, densities, Mach values, distances, pressures, and, speeds. These modules may be used to implement types with restricted ranges or special interpretations (e.g., angle is used to represent latitude).

C:3.1.2 SHARED DATA TYPE MODULE

The shared data types are data types that are used in a few modules. For example, the indicator control data type is introduced in the Device Interface Module in order to communicate the three possible states of an indicator (on, off and blinking) to several virtual devices.

C:3.1.3 LOCAL DATA TYPE MODULE

The local data types are the data types that were introduced in the interface definitions of another module.

C:3.2 PHYSICAL MODEL MODULE DECOMPOSITION

The Physcical Model Module comprises the seven modules described below.

C:3.2.1 EARTH MODEL MODULE

The Earth Model Module hides models of the earth and its atmosphere. This set of models includes models of local gravity, the curvature of the earth, pressure at sea level, magnetic variation, the local terrain, the rotation of the earth, coriolis force, and atmospheric density.

C:3.2.2 AIRCRAFT MOTION MODULE

The Aircraft Motion Module hides models of the aircraft's motion. They are used to calculate aircraft position, velocity and attitude from observable inputs.

C:3.2.3 SPATIAL RELATIONS MODULE

The Spatial Relations Module contains models of three-dimensional space. These models are used to perform coordinate transformations as well as angle and distance calculations.

C:3.2.4 HUMAN FACTORS MODULE

The Human Factors Module is based on models of pilot reaction time and perception of simulated continuous motion. The models determine the update frequency appropriate for symbols on a display.

C:3.2.5 WEAPON BEHAVIOR MODULE

The Weapon Behavior Module contains models used to predict weapon behavior after release.

C:3.2.6 TARGET BEHAVIOR MODULE

The Target Behavior Module contains models used to predict target behavior, such as whether it is stationary or moving.

C:3.2.7 FILTER BEHAVIOR MODULE

The Filter Behavior Module contains digital models of physical filters. They can be used by other programs to filter potentially noisy data.

C:3.3 DATA BANKER MODULE

The decomposition of the Data Banker into submodules is hidden from user programs, which should not be sensitive to changes in its internal structure.

C:3.4 SYSTEM GENERATION MODULE DECOMPOSITION

C:3.4.1 SYSTEM GENERATION PARAMETER MODULE

The System Generation Parameter Module provides values for all the system generation parameters defined in other modules, including those specified in module interfaces and those defined in the module implementations. There is a submodule of the System Generation Parameter Module for each module in the rest of the system; each of these submodules is in turn composed of an external parameter submodule and an internal parameter submodule. External parameters of a module are available to other modules; internal parameters are secrets of the module. The primary secrets of this module are the values of the parameters for a particular version of the system.

C:3.4.2 MODULE VERSION SELECTION MODULE

This module stores alternative implementations of each module. It allows a user to indicate which alternative(s) should be chosen for each module. If the module implements an abstract data type, a different alternative may be specified for each variable of that type. The secret of this module is the library structure used to store the various versions of the modules and the procedures for inserting the selected implementation in the code.

C:3.4.3 SUBSET SELECTION MODULE

This module selects subsets of each module in order to assemble a desired subset version of the system. Its primary secret is the "Uses" relation; the secondary secrets are the representation of the relation and the algorithms used to select the programs that will be included in the resulting system.

C:3.4.4 SUPPORT SOFTWARE MODULE

The Support Software Module includes additional software required to generate and check out a running system, including a macroprocessor, a simulator, an assembler, and communications software.

C:3.5 SOFTWARE UTILITY MODULE

This module provides common software services needed by other modules. The primary secrets are the algorithms that use of Extended Computer and Application Data Type facilities to start and stop a process, signal a value change, and check to make sure a value is between prescribed bounds. A service is also provided to signal when the conjunction or disjunction of two conditions becomes true or false, given the adjunct conditions.

Mathematical services include exponentiation, logarithm, maximum, minimum, and absolute value functions.

This module will be expected to grow as we recognize useful utilities. Programmers are required to submit descriptions of useful utilities for inclusion in this module as a change to this document. However, the complete list of approved utilities will be maintained as an appendix to this document.

C:3.6 RESOURCE MONITOR MODULE

A resource monitor module will be designed for every resource (data structure or device) that will be accessed by more than one process. The monitor will enforce the protocol that has been adopted for shared use of that resource. All accesses to the shared resource will be made by calls to monitor access routines.

Acknowledgements

P. C. Clements' efforts have led to substantial improvements in the structure described in this document and to substantial clarifications in the writing. Comments on earlier drafts by B. Amlicke, S. Bouchard, L. Chmura, H. Elovitz, C. Heitmeyer, R. Krutar, D. Weiss, and J. Shore were very helpful.

IV. REFERENCES

[1] Heninger, K., Kallander, J., Parnas, D., and Shore, J.; Software Requirements for the A-7E Aircraft; NRL Memorandum Report 3876; 27 November 1978.

[2] Parnas, D.L.; "On the Criteria To Be Used in Decomposing Systems into Modules"; Comm. ACM, Vol. 15, No. 12 (December 1972), pp. 1053-1058.

[3] Parker, A., Heninger, K., Parnas, D., and Shore, J.; Abstract Interface Specifications for the A-7E Device Interface Module; NRL Memorandum Report 4385, 20 November 1980.

[4] Heninger, K., and Parnas, D.L.; Interface Specifications for the A-7 Extended Computer Module; in progress.

[5] Mode Determination documentation, to be published.

[6] Data Banker Documentation, to be published.

[7] Process Structure Documentation, to be published.

[8] Parnas, D.L.; "Designing Software For Extention and Contraction", Proceedings of the 3rd International Conference on Software Engineering (10-12 May 1978), pp. 264-277.

[9] Parnas, D.L., and Wuerges, H.; "Response to Undesired Events in Software Systems"; Proc. Second Int. Conf. Software Eng., pp. 437-446; 1976.

[10] Clements, P.C.; Function Specifications for the A-7E Function Driver Module; NRL Memorandum Report 4658, October 1981.

[11] Clements, P.C.; Interface Specifications for the A-7E Shared Services Module; draft in progress, 22 June 1981.

[12] Clements, P.C.; Abstract Interface Specifications for the A-7E Application Data Type Module; draft in progress.

[13] Clements, P.C., and Parker, A.; Abstract Interface Specifications for the A-7E Physical Models Module; draft in progress.

[14] System Generation Module documentation, to be published.

[15] Uses hierarchy documentation, to be published.

V. GLOSSARY

abstract interface	an abstraction that represents more than one interface (see interface, module interface); it consists of the assumptions that are included in all of the interfaces that it represents.
abstraction	a description of a set of objects that applies equally well to any one of them. Each object is an instance of the abstraction.
access function	see <u>access program</u>.
access program	a program that may be called by programs outside of the module to which it belongs. Most run-time communication between modules is effected by invocation of access programs. There are several different sorts of access functions: some return information to the caller, some change the state of the module to which they belong, and some do both.
consumer	a program that requires data produced by another program.
event	(1) change in a condition; (2) signal from a module to its user programs indicating the occurrence of some change within the module. Events resemble hardware interrupts because they occur at unpredictable times and are not synchronized with the control flow of user programs. In the A-7E program, events will be signalled to user programs using an event mechanism (see below).
event mechanism	a programming construct that allows processes to communicate about the occurrence of events. Typical operations provided include an operation to wait for a particular event to occur and an operation to signal that a particular event has occurred.
hidden submodule	a submodule whose existence is part of the secret of the parent module.
interface	(1) between two programs: the assumptions that each programmer needs to make about the other program in order to demonstrate the correctness of his own program.
	(2) between a program and a device: assumptions about the device that must be accounted for in the program in order for the program to work as expected.

internal program	a program that is not accessible to programs outside the module; the existence of the internal program is part of the secret of the module.
module	a programming work assignment consisting of one or more programs. A module may be divided into smaller modules (submodules).
module facility	the access programs and events provided by a module in order to allow user programs to be independent of the module secret. A complete description of a facility is a specification of the module.
module hierarchy	a hierarchy defined by the relation "contains" on pairs of modules.
module implementation	the algorithms, data structures, and programs that satisfy the module specification.
module interface	the set of assumptions that the authors of external programs may make about the module. It includes restrictions on the way that the module may be used. In the A-7E software, modules communicate either by one module using access programs from the other module, or by one module being notified of an event that was signalled by the other module. The interface consists of assumptions about the availability of the access programs, the syntax of the calls on the access programs, the behavior of the access programs, and the meaning of events. See also interface.
module secret	see secret, module implementation
module specification	a description of a module interface; see also module facility.
module's structure	the way that a software module is divided into submodules and programs
primary secret	the characteristics other than decisions by the module designer that a module is intended to hide. See also secondary secret

process	a subset of the run-time events of the system used as administrative units in the run-time allocation of processors. See also process definition .
process definition	the program that controls the sequence of actions by a process.
producer	a module that provides data for use by other programs. A program that returns to its caller an output parameter that is a function of the input that the caller provided is not considered a data producer. Examples of data producers include programs that read values measured by sensors, or calculate physical characteristics based on mathematical models.
program	a named, machine executable, description of an algorithm. The name may be used to invoke the program's execution. A program may include a description (declaration) of the data structures that it uses; it may invoke other programs and refer to data structures that have been described in other programs. See also subprogram, process definition.
required function	used in the sense of [1]. Each requirements function is the determination of the value of a specific, closely related set of output values. The system performs all of its requirements functions when it properly determines the value of all of its outputs.
secondary secret	software design decisions made to implement the abstraction that hides the primary secret.
secret	the facts about the module that are <u>not</u> included in its interface; i.e., assumptions that user programs are not allowed to make about the module. The correctness of programs in other modules must not depend on those facts. The secrets tell how the module's specification has been satisfied. See also module specification, primary secret, secondary secret.
submodule	any module that is a component of a higher level module.
subprogram	a subprogram is a program that can be invoked by another program. A subprogram may be either a subroutine or a macro.
sysgen parameters	a symbol used as a placeholder for values that will be supplied just before a system is generated.

undesired event (UE)	a run-time event that the designers hope will not occur. Production versions of the A-7E program are written on the assumption that they do not occur.
undesired event assumption	assumptions about what constitutes improper use of a module by user programs, e.g., calling an access program with parameters of the wrong type.
use, uses	Program A uses program B if there must be a correct version of B present for A to run correctly. A program uses a module if it uses at least one program from that module. A module uses another module if at least one program uses that module.
user programs	all programs that use programs from a module but are not part of that module. The term "user" is relative to the module being discussed.
virtual computer	a computer-like set of instructions implemented, at least in part, by software.
virtual machine	see virtual computer.
visible submodules	submodules whose existence is visible to user programs.